CSC401 Tutorial 3 - Regular Expressions

Zhewei Sun

This is a quick introduction to regular expressions to get you up to speed on Assignment 1. For pre-processing and feature extraction, you should try to use regular expressions instead of doing brute-force string processing.

In [1]: sentence = "CSC 401 Tutorial 3: reeeeg. eeexp. yeeah"

Regular Expressions in Python

re (https://docs.python.org/3/library/re.html#re.sub) is the built-in regular expression library for Python. As we'll see later, it has many features beyond search.

In [2]: import re

re.compile builds a regex pattern that can be reused throughout your program. Note the letter r in front of the string. It tells Python to read the string as a raw string so control characters like \b can be properly captured.

In [3]: pattern = re.compile(r"ee")
   pattern_num = re.compile(r"\d+")

Some useful regex operations you can use:

re.search finds first match in the string:

In [4]: pattern.search(sentence)
Out[4]:<_sre.SRE_Match object; span=(21, 23), match='ee'>
In [5]: pattern.search("No Match")
re.findall finds all occurrences of a pattern:

```
In [6]: pattern.findall(sentence)
Out[6]: ['ee', 'ee', 'ee']
```

Note that for overlapping patterns, it retrieves the left-most one.

```
In [7]: pattern_num.findall(sentence)
Out[7]: ['401', '3']

In [8]: pattern_num.findall(sentence, 10)
Out[8]: ['3']

In [9]: pattern.findall(sentence, 10, 15)
Out[9]: []
```

re.sub substitutes the matches in re.findall to the string of your choice:

```
In [10]: pattern.sub('zz', sentence)
Out[10]: 'CSC 401 Tutorial 3: rzzeg. zzexp. yzzah'

In [11]: pattern.sub('', sentence, count=2)
Out[11]: 'CSC 401 Tutorial 3: reg. exp. yeeah'
```

You can also do something fancy with the replacement:

```
In [12]: def fun(x):
   # x will be a Match object like the one returned in re.search()
   if int(x.group()) > 100:
       return 'bignum'
   return 'smallnum'
pattern_num.sub(fun, sentence)
```

```
Out[12]: 'CSC bignum Tutorial smallnum: reeeeg. eeexp. yeeah'
```

RegEx Patterns
Basic Patterns

. matches any character

In [13]: re.compile(r"." ).findall(sentence)

Out[13]: ['CSC', '40', '1 T', 'uto', 'ria', 'l 3', ': r', 'eee', 'g.', 'eee', 'xp.', 'ye', 'eah']

\w matches any alphanumeric characters, plus underscore:

In [14]: re.compile(r"\w\w").findall(sentence)

Out[14]: ['CS', '40', 'Tu', 'to', 'ri', 'al', 're', 'ee', 'ee', 'ex', 'ye', 'ea']

\W matches all non-alphanumeric characters:

In [15]: re.compile(r"\W").findall(sentence)

Out[15]: [' ', ' ', ' ', ':', ' ', '.', ' ', '.', ' ']

\s matches all whitespace characters:

In [16]: re.compile(r"\s").findall(sentence+' \n')

Out[16]: [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' \n']

\S does the opposite:
\d matches all numerical characters, \D does the opposite

Number Patterns

Sometimes you want to match consecutive characters of interest, these will come handy
+ matches 1 or more

* matches 0 or more

? matches 0 or 1

Some examples:

All sequences of consecutive e's:

In [19]: \texttt{re.compile(r"e+").findall(sentence)}
Out[19]: ['eee', 'eee', 'ee']

Words containing the letter r:

In [20]: \texttt{re.compile(r"\w*r+\w*").findall(sentence)}
Out[20]: ['Tutorial', 'reeeg']

Tokens with exactly one a:

In [21]: \texttt{re.compile(r" +a? +").findall(" a aa aaaa ")}
Out[21]: [' a ']

**Boundary Patterns**

Patterns for string and word boundaries:

^ - beginning of string

$ - end of string

\b - word boundry (beginning or end)

\B - non-word boundry

In [22]: \texttt{re.compile(r"^..").findall(sentence)}
Out[22]: ['CS']
In [23]: re.compile(r"..$" ).findall(sentence)
Out[23]: ['ah']

In [24]: re.compile(r"^..$" ).findall(sentence)
Out[24]: []

Be careful that \b does not consider puncuations. It only considers consecutive \w strings to be a word:

In [25]: re.compile(r"\b\w+\b" ).findall(sentence)
Out[25]: ['CSC', '401', 'Tutorial', '3', 'reeeg', 'eeexp', 'yeeah']

Words containing eee that is not at the beginning:

In [26]: re.compile(r"\b\w*\Beee\w*\b" ).findall(sentence)
Out[26]: ['reeeg']

If we remove the \B, eeexp will also be matched:

In [27]: re.compile(r"\b\w*\Beee\w*\b" ).findall(sentence)
Out[27]: ['reeeg', 'eeexp']

More Number Patterns

The {} clause allows you to specify the number:

{2} - exactly two copies

{2,} - two or more copies

{2,3} - two or three copies

In [28]: re.compile(r"\ba{2}\b" ).findall(" a aa aaa aaaa ")
Out[28]: ['aa']

In [29]: re.compile(r"\ba{2,}\b" ).findall(" a aa aaa aaaa ")
Out[29]: ['aa', 'aaa', 'aaaa']
In [30]: re.compile(r"\ba{2,3}\b").findall(" a aa aaa aaaa ")
Out[30]: ['aa', 'aaa']

**Character Classes**

A character class can be created using square brackets. The regex will match any of the characters within.

Match words that begin with either r or e:

In [31]: re.compile(r"\b\[re]\w*\b").findall(sentence)
Out[31]: ['reeeg', 'eeexp']

You can also opt to NOT match a class by adding ^:

In [32]: re.compile(r"\b[^re]\w*\b").findall(sentence)
Out[32]: ['CSC', ' 401', ' Tutorial', ' 3', 'yeeah']

Some shorthands for character classes:

- [0-9] - numbers
- [a-z] - lowercase alphabets
- [A-Z] - uppercase alphabets

In [33]: re.compile(r"\b[A-Z]+\b").findall(sentence)
Out[33]: ['CSC']

These classes can also be combined:

In [34]: re.compile(r"\b[0-9a-z]+\b").findall(sentence)
Out[34]: ['401', '3', 'reeeg', 'eeexp', 'yeeah']

**OR Clauses**
Use (?: | ) to create OR clauses. ?: tells Python that the content of this parenthesis should not be used as a group.

```python
In [35]: re.compile(r"\b(?:[a-z]|[A-Z])+\b").findall(sentence)
Out[35]: ['CSC', 'Tutorial', 'reeeg', 'eeexp', 'yeeah']
```

```python
In [37]: re.compile(r"\b(?:r|e)\w*\b").findall(sentence)
Out[37]: ['reeeg', 'eeexp']
```

**Lookaheads**

Sometimes we want to match something but don't want them to be retrieved. For example:

```python
In [38]: re.compile(r" a+ ").findall(" a aa aaa aaaa ")
Out[38]: [' a ', ' aaa ']
```

We can use lookaheads in this case:

(?) - lookahead

(?)= - lookbehind <-- requires simple fixed length patterns

```python
In [39]: re.compile(r"(?<= )a+(?= )").findall(" a aa aaa aaaa ")
Out[39]: ['a', 'aa', 'aaa', 'aaaa']
```

**Examples**

Now we can do something useful, say extracting tokens containing three or more e's:

```python
In [41]: re.compile(r"\S*e{3}\S*").findall(sentence)
Out[41]: ['reeeg.', 'eeexp.']
```

Tokens that begin with a number:
Tokens with at least two numerical characters:

In [42]: re.compile(r"\S*\d\S*\d\S*" ).findall(sentence)
Out[42]: ['401', '3: ']

Finding tags

In [45]: re.compile(r"<[^>]+>" ).findall("<title>Hello world<\/title>" )
Out[45]: ['<title>', '<\title>']

Tokens with only lower case letters:

In [47]: re.compile(r"(?::|(?<=\s))[a-z]+(?=\s|$)" ).findall(sentence)
Out[47]: ['yeeah']